Conjunctive Context-Free Path Queries

Jelle Hellings
Hasselt University and transnational University of Limburg
jelle.hellings@uhasselt.be

ABSTRACT

In graph query languages, regular expressions are commonly used to specify the labeling of paths. A natural step in increasing the expressive power of these query languages is replacing regular expressions by context-free grammars. With the Conjunctive Context-Free Path Queries (CCFPQ) we introduce such a language based on the well-known Conjunctive Regular Path Queries (CRPQ).

First, we show that query evaluation of CCFPQ has polynomial time data complexity. Secondly, we look at the generalization of regular expressions, as used in CRPQ, to regular relations and show how similar generalizations can be applied to context-free grammars, as used in CCFPQ. Thirdly, we investigate the relations between the expressive power of CRPQ, CCFPQ, and their generalizations. In several cases we show that replacing regular expressions by context-free grammars does increase expressive power. Finally, we look at including context-free grammars in more powerful logics than conjunctive queries. We do so by adding negation and provide expressivity relations between the obtained languages.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query languages*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*

General Terms

Theory

1. INTRODUCTION

Central in graph structured data are data-elements (nodes), direct relations between data-elements (edges), and indirect relations between data-elements (paths). Due to its intuitive nature, the graph structured data model has a broad range of applications, including applications in the social sciences, bio-informatics, the semantic web, geographical data, process modeling, and in formal verification.

Usually, graph-structured data is queried based on the paths in the data. A common way to specify these paths is by specifying the path labeling using regular expressions over edge labels [24]. An example of a query language using such regular expressions is CRPQ [11, 12]. To increase the expressive power of CRPQ, the *Extended Conjunctive Regular Path Queries* (ECRPQ) [5] have been proposed. In ECRPQ, the regular expressions are generalized to regular relations. These regular relations are regular expressions over tuples of edge labels, and they specify the labeling of tuples of paths. This allows ECRPQ to compare distinct paths.

Both CRPQ and ECRPQ have efficient query evaluation with polynomial time complexity in terms of the size of the queried graph. Furthermore, widespread usage of regular expressions in common tasks, such as editing, programming, and system administration, reduces the learning curve for working with regular expressions in graph query languages. Besides the practical motivations for using regular expressions, there are also theoretical motivations. Regular expressions are well-understood and have a strong foundation in formal languages and automata theory [20].

In search of more expressive variants of CRPQ one can also consider an alternative route, namely replacing regular expressions by more powerful formal grammars. In string recognition, parsing, and in compiler construction the expressive power of regular expressions is considered to be insufficient. As such, the usage of the more expressive context-free grammars is widespread [14]. Due to this widespread usage, there are many reasonably efficient string recognition algorithms for context-free grammars. Practical string recognizing algorithms for context-free grammars have a cubic running time with respect to the length of the string [10, 16, 29, 30, 35] although slightly better algorithms are known [31].

Increasing the expressive power of path based query languages by using context-free grammars, instead of regular expressions, is thus a logical step. Moreover, initial steps in the usage of context-free grammars as graph query languages have already been made in the context of model checking [18] and in bio-informatics [28]. These initial steps provide context-free recognition algorithms for graphs with running times of $\mathcal{O}(|G|\cdot n^5)$ [18] and $\mathcal{O}(n^3m^2)$ [28], where $|G|$ is dependent on the size of the grammar, n is the number of

10.5441/002/icdt.2014.15

nodes in the graph, and m is the maximum path length. Furthermore, Lange [18] proposes to investigate if graph query languages using richer, mildly context-sensitive, grammar formalisms are feasible. Examples of such grammar formalisms are the alternating context-free grammars [25] and the conjunctive grammars [26].

Motivated by, on the one hand, the desire to extend the expressive power of CRPQ, and, on the other hand, existing applications of context-free grammar-based graph query languages, we propose the *Conjunctive Context-Free Path Queries* (CCFPQ). CCFPQ is the query language obtained by replacing the usage of regular expressions to specify the labeling of paths in CRPQ by context-free grammars. We illustrate the power of CCFPQ with a straightforward example.

Example 1. Consider a family tree with *parent* and *child* edges. The context-free grammar

$$\mathtt{N} \to \mathit{parent}\,\mathit{child}, \quad \mathtt{N} \to \mathit{parent}\,\mathtt{N}\,\mathit{child},$$

which is not regular, recognizes paths with labeling of the form $\mathit{parent}^n\,\mathit{child}^n$ $(1 \leq n)$. Now the CCFPQ query

$$\mathrm{Q}(d_1, d_2) \leftarrow \mathtt{N}(d_1, d_2)$$

gives us pairs of n-th generation descendants of a common ancestor. This query Q is not expressible by CRPQ.

Our main result is that CCFPQ has efficient query evaluation, while at the same time having greater expressive power than CRPQ. For query evaluation we provide a context-free recognition algorithm for graphs. Our algorithm has a running time of $\mathcal{O}((me) + (mn)^3)$, where m is the number of non-terminals in the grammar, e is the number of edges in the graph, and n is the number of nodes in the graph. This improves on earlier graph recognition algorithms for context-free grammars.

We show an important restriction in the usage of path variables: freely using path variables allows us to query for paths whose traces match the intersection of two context-free languages. In this case, we prove that query evaluation is undecidable, even for a fixed query. These results also negatively answer an open question from Lange [18]: query evaluation is undecidable for graph query languages that use alternating context-free grammars [25] or conjunctive grammars [26] to specify the labeling of paths.

We also look at the generalization of regular expressions, as used in CRPQ, to regular relations, resulting in ECRPQ. We show how similar generalizations can be applied to context-free grammars, as used in CCFPQ. We include a broad study of the expressive power of CRPQ, CCFPQ, and the introduced generalizations.

We show that, in general, CCFPQ can express more queries than CRPQ. We also show that CRPQ and CCFPQ have equivalent expressive power on unlabeled graphs. Interestingly, this collapse also holds for the respective generalizations of CRPQ and CCFPQ with grammars over tuples of edge labels. We also introduce context-free grammars in more powerful logics by adding negation to CCFPQ, and we provide a strict hierarchy in expressive power of the resulting query languages.

Organization. The next section introduces the necessary preliminaries. Section 3 introduces CCFPQ, together with a query evaluation algorithm. Section 3 also discusses the introduction of path variables, the inclusion of paths in the query result, and the introduction of generalized grammars over tuples of edge labels. Section 4 studies the differences in expressive power of CRPQ, CCFPQ, and their generalizations with grammars over tuples of edge labels. Section 5 looks at the expressive power of context-free grammars in more powerful logics. Section 6 concludes and looks at future work.

2. PRELIMINARIES

Let $\mathcal{A}$ be a finite set of edge labels. An *edge-labeled directed graph* is a tuple $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{A} \times \mathcal{V}$ a directed edge-relation. A *path* $\pi = n_1 e_1 \dots n_{i-1} e_{i-1} n_i$ in graph $\mathcal{D}$ is a non-empty finite sequence of nodes connected by edges e_j, for $1 \leq j < i$, which satisfy $e_j = (n_j, l_j, n_{j+1})$. The length of path π is defined by $|\pi| = i$. We write $n_1 \pi n_i$ to indicate that path π starts at node n_1 and ends at node n_i. The *trace* of path π is defined by $\mathcal{T}(\pi) = l_1 \dots l_i$, we remark that traces are strings over the edge labels $\mathcal{A}$ and that the trace of a node is the empty string. The set of traces between nodes n and m is defined by $\mathcal{T}_{\mathcal{S}}(n, m) = \{\mathcal{T}(\pi) \mid n \pi m\}$.

We denote the *concatenation* of two strings s_1 and s_2 by $s_1 + s_2$. We denote the *concatenation* of two paths $a\pi_1 b$ and $b\pi_2 c$ by $\pi_1 + \pi_2$. We only concatenate paths if the last node on the first path π_1 is equal to the first node on the second path π_2.

A *context-free grammar* is a 3-tuple $G = (N, A, P)$ where N is a finite set of non-terminals, A is a finite set of symbols which we call the alphabet, and P is a finite set of production rules. Usually, the alphabet A is a subset of the edge-labels $\mathcal{A}$. Each production rule is of the form $\mathtt{S} \to m$ where m is a finite string over $N \cup A$. We write $\mathtt{S} \to \lambda$ when $|m| = 0$.[1]

The production rules describe rewrites of strings over $N \cup A$ allowed by the grammar. If we have a string $s = s_1 \dots s_j$ with $s_k = \mathtt{N}$, for $1 \leq k \leq j$, $\mathtt{N} \in N$, and production rule $\mathtt{N} \to m_1 \dots m_i$, then we can rewrite s into

$$s_1 \dots s_{k-1} m_1 \dots m_i s_{k+1} \dots s_j.$$

We write $s \to_G q$ for strings s and q whenever s can be rewritten into q by a finite number of rewrites using the production rules in G. The *language* of grammar $G = (N, A, P)$ with respect to start non-terminal $\mathtt{S} \in N$ is defined by $\mathcal{L}(G_{\mathtt{S}}) = \{s_1 \dots s_j \text{ a finite string over } A \mid \mathtt{S} \to_G s_1 \dots s_j\}$.

To simplify working with context-free grammars we assume that all context-free grammars are in the following normal form:

[1]We deviate from the usual definition of a context-free grammar by not including a special start non-terminal. We will specify the start non-terminal in the queries using context-free grammars.

Definition 1. A context-free grammar is in *Normal Form* if all productions are of the form $\mathtt{A} \to \mathtt{B}\,\mathtt{C}$, $\mathtt{A} \to \sigma$, or $\mathtt{A} \to \lambda$ where $\mathtt{A}, \mathtt{B}, \mathtt{C} \in N$ and $\sigma \in A$.

By allowing arbitrary production rules of the form $\mathtt{A} \to \lambda$, this normal form deviates from the usual *Chomsky Normal Form* [20]. This is a consequence of the lack of start non-terminals in our grammar definition. Using standard techniques [20] one can show:

PROPERTY 1. *For every $G = (N, A, P)$ there exists a grammar $G' = (N', A, P')$ with G' in normal form and, for every $\mathtt{N} \in N$, $\mathcal{L}(G_{\mathtt{N}}) = \mathcal{L}(G'_{\mathtt{N}})$. G' can be constructed from G in polynomial time and size with respect to $|N|$ and the sum of the lengths of all production rules.*

Let L be a query language and M be a data model. The *query evaluation problem* has as input a database D from M, query Q from L, and tuple t and decides if t is in the query result of Q on D. In the framework introduced by Vardi [32], the *data complexity* of the query evaluation problem refers to the complexity with respect to the size of database D, given a fixed query Q, and the *combined complexity* of the query evaluation problem refers to the complexity with respect to the size of query Q and database D.

3. CONJUNCTIVE CONTEXT-FREE PATH QUERIES

We can apply context-free grammars on graphs in a similar way as regular expressions are applied on graphs by CRPQ. Given a graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ and grammar $G = (N, A, P)$, we define *context-free relations* $\mathcal{R}_{\mathtt{N}} \subseteq \mathcal{V} \times \mathcal{V}$, for every $\mathtt{N} \in N$, such that $\mathcal{R}_{\mathtt{N}} = \{(n, m) \mid \exists n\pi m\ (\mathcal{T}(\pi) \in \mathcal{L}(G_{\mathtt{N}}))\}$. We use the notation $\exists n\pi m\, \varphi$ for node variables n and m as a shorthand for $\exists \pi\ (n\pi m \wedge \varphi)$. Using this interpretation of non-terminals over graphs, we define the *Conjunctive Context-Free Path Queries* as follows:

Definition 2. Let $G = (N, A, P)$ be a grammar. A *Conjunctive Context-Free Path Query* (CCFPQ) over grammar G is an expressions of the form:

$$\mathrm{Q}(\vec{\nu}) \leftarrow \exists \vec{\mu} \bigwedge_{i \in \mathcal{I}} \mathtt{N}_i(n_i, m_i),$$

where Q is the name of the query, $\vec{\nu}$ is a tuple of node variables, $\vec{\mu}$ is a tuple of distinct node variables that do not occur in $\vec{\nu}$, i ranges over a finite index set $\mathcal{I}$, $\mathtt{N}_i \in N$ is a non-terminal, and n_i and m_i are node variables taken from $\vec{\nu}$ or $\vec{\mu}$. A CCFPQ built over a regular grammar is a *Conjunctive Regular Path Query* (CRPQ).

The semantic evaluation of query Q on graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, denoted by $\mathrm{Q}_{\mathcal{D}}$, is defined by the usual semantics of first-order queries over $(\mathcal{V}; \mathcal{R}_{\mathtt{N}_i}, i \in \mathcal{I})$, whereby atoms $\mathtt{N}_i(n_i, m_i)$ are interpreted as the relations $\mathcal{R}_{\mathtt{N}_i}(n_i, m_i)$.

Query $\mathrm{Q}(\vec{\nu})$ is a *boolean* query whenever $\vec{\nu}$ is the empty tuple $\langle\rangle$. The evaluation of a boolean query over graph $\mathcal{D}$ is interpreted in the usual way: $\mathrm{Q}_{\mathcal{D}} = \emptyset$ is interpreted as false, and $\mathrm{Q}_{\mathcal{D}} = \{\langle\rangle\}$ is interpreted as true.

Example 2. Consider the grammar $G = (N, A, P)$ with $N = \{\mathtt{N}\}$, $A = \{\uparrow, \downarrow\}$, and $P = \{\mathtt{N} \to \mathtt{N}\,\mathtt{N}, \mathtt{N} \to \uparrow \mathtt{N} \downarrow, \mathtt{N} \to \lambda\}$. The grammar describes properly nested open-parentheses $\uparrow$ and close-parentheses $\downarrow$.

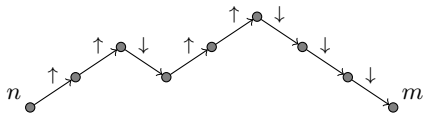

Figure 1: An example of a zigzag-graph between the nodes indicated with n and m.

The query $\mathrm{Q}_{zigzag}(n_1, n_2) \leftarrow \mathtt{N}(n_1, n_2)$ returns pairs of nodes if they are connected by a path of properly nested $\uparrow$ and $\downarrow$ parentheses. The result of Q_{zigzag} on the zigzag-graph in Figure 1 will thus include the node pair (n, m), but not the node pair (m, n).

In general, we only specify the relevant production rules as the set of non-terminals and the alphabet are implied by the production rules. Usually, we write regular expressions in CRPQs instead of using non-terminal atoms from corresponding regular grammars.

Example 3. Consider the regular grammar

$$\mathtt{TC} \to \sigma, \quad \mathtt{TC} \to \sigma\,\mathtt{TC}\ (\forall \sigma \in A).$$

The query $\mathrm{Q}_{\mathtt{TC}}(n, m) \leftarrow \mathtt{TC}(n, m)$ evaluates to the transitive closure of a graph and the boolean query

$$\mathrm{Q}_{cyclic}() \leftarrow \exists n\ \mathtt{TC}(n, n)$$

evaluates to true if and only if the graph is cyclic.

3.1 Query evaluation

Let $G = (N, A, P)$ be a grammar and $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ be a graph. We consider queries $\mathrm{Q}(\vec{\nu})$ build over $\mathtt{N} \in N$. If we have the relations $\mathcal{R}_{\mathtt{N}}$, then we can reduce query evaluation for CCFPQ to query evaluation of conjunctive first-order queries. For calculating the relations $\mathcal{R}_{\mathtt{N}}$, we present Algorithm 1.

Algorithm 1 Context-free recognizer for graphs.

Input: Edge-labeled directed graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$
Context-free grammar $G = (N, A, P)$ in Normal Form
Output: $\{(\mathtt{N}, n, m) \mid (n, m) \in \mathcal{R}_{\mathtt{N}}\}$

```
1: r := {(N, n, n) | (n ∈ V) ∧ (N → λ ∈ P)} ∪
        {(N, n, m) | ((n, l, m) ∈ E) ∧ (N → l ∈ P)}
2: new := r
3: while new ≠ ∅ do
4:    pick and remove a (N, n, m) from new
5:    for all (M, n', n) ∈ r do
6:       for all (N' → M N ∈ P) ∧ ((N', n', m) ∉ r) do
7:          new := new ∪ {(N', n', m)}
8:          r := r ∪ {(N', n', m)}
9:    for all (M, m, m') ∈ r do
10:      for all (M' → N M ∈ P) ∧ ((M', n, m') ∉ r) do
11:         new := new ∪ {(M', n, m')}
12:         r := r ∪ {(M', n, m')}
13: return r
```

Algorithm 1 parses the graph $\mathcal{D}$ using the context-free grammar G in a bottom-up way, thereby basically applying CYK-parsing [14] on graphs.

PROPOSITION 1. *Let $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ be a graph and let $G = (N, A, P)$ be a grammar. Let r be the output of Algorithm 1 on $\mathcal{D}$ and G. We have $(\mathtt{N}, n, m) \in r$ if and only if $(n, m) \in \mathcal{R}_{\mathtt{N}}$.*

PROOF. We have $(n, m) \in \mathcal{R}_{\mathtt{N}}$ if and only if there exists a path $n\pi m$ with $\mathtt{N} \to_G \mathcal{T}(\pi)$. Using induction on the number of rewrite steps from $\mathtt{N}$ to $\mathcal{T}(\pi)$, we prove $(\mathtt{N}, n, m) \in r$ if $(n, m) \in \mathcal{R}_{\mathtt{N}}$.

We have one rewrite step if and only if there is a production rule of the form $\mathtt{N} \to a$ with $a \in A \cup \{\lambda\}$. In this case, we have $(\mathtt{N}, n, m) \in r$ by line 1. Now assume when $\mathtt{N'} \to_G \mathcal{T}(n'\pi'm')$ in at most $1 \leq i$ rewrite steps, then $(\mathtt{N'}, n', m') \in r$. Let $\mathtt{N} \to_G \mathcal{T}(n\pi m)$ in $i + 1$ rewrite steps. The first production rule applied must be of the form $\mathtt{N} \to \mathtt{A}\,\mathtt{B}$ for $\mathtt{A}, \mathtt{B} \in N$. Let $n\pi m = \pi_1 +\!\!+\, \pi_2$ with $\mathtt{A} \to_G \mathcal{T}(n\pi_1 c)$ and $\mathtt{B} \to_G \mathcal{T}(c\pi_2 m)$. These rewrites happen in at most i rewrite steps, we thus have $(\mathtt{A}, n, c), (\mathtt{B}, c, m) \in r$. Hence, by the inner loops at lines 5 and 9, we have $(\mathtt{N}, n, m) \in r$.

The only if direction follows trivially from the algorithm as $(\mathtt{N}, n, m)$ is only added to r whenever a suiting production rule is found, resulting in a proof for $(n, m) \in \mathcal{R}_{\mathtt{N}}$. □

The complexity of query evaluation of CCFPQ is determined by the complexity of Algorithm 1 and the complexity of query evaluation of conjunctive first-order queries. We are interested in data and combined complexity.

THEOREM 1. *Let $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ be a graph and let $G = (N, A, P)$ be a grammar. Algorithm 1 applied to $\mathcal{D}$ and G has a worst-case complexity of $\mathcal{O}(|N||\mathcal{E}| + (|N||\mathcal{V}|)^3)$.*

By Theorem 1, we conclude that Algorithm 1 is an improvement on the asymptotic polynomial running time (with respect to the size of the graph) of the algorithm provided by Lange [18].

For the conjunctive first-order queries, the query evaluation problem has efficient solutions with polynomial time data complexity and NP-complete combined complexity [1]. Using Theorem 1, we extend these results to CCFPQ.

COROLLARY 1. *For CCFPQ queries, the query evaluation problem has*

1. *Polynomial time data complexity,*
2. *NP-complete combined complexity.*

PROOF (SKETCH). Algorithm 1 reduces query evaluation of CCFPQ to query evaluation of conjunctive queries and does so in polynomial time (with respect to the size of the graph and grammar). NP-hardness of combined complexity of the query evaluation problem follows from NP-hardness of the query evaluation problem for conjunctive queries and CRPQ [1, 5]. □

3.2 Explicit path variables

Variants of CRPQ use explicit path variables [5]. In these variants, path variables are used in the form

$$\mathrm{Q}() \leftarrow \exists nm\ \exists n\pi m\ r_1(\pi) \wedge r_2(\pi),$$

where the regular expressions r_1 and r_2 both place conditions on the trace of a single path. The regular expressions are closed under intersection. Hence, indirectly all representations of CRPQ support path variables [13]. For CRPQ, path variables thus do not have an impact on the expressive power, omitting path variables can however influence the succinctness of the query language.

The context-free grammars are not closed under intersection. Hence, adding path variables, such that several nonterminals can put restrictions on a single path, would be a logical extension of CCFPQ.

THEOREM 2. *Consider CCFPQ queries using path variables of the form $\mathrm{Q}() \leftarrow \exists nm\ \exists n\pi m\ \mathtt{M}(\pi) \wedge \mathtt{M}(\pi)$.*

1. *There is a fixed graph $\mathcal{D}$ such that the query evaluation problem is undecidable.*
2. *There is a fixed query Q such that the query evaluation problem of Q is undecidable.*

PROOF (SKETCH). For proving 1. it is well-known that the emptiness of the intersection of two context-free languages is undecidable. We can easily construct a graph $\mathcal{D}$ that accepts all possible strings. Now $\langle\rangle \in \mathrm{Q}_{\mathcal{D}}$ if and only if the intersection of $\mathtt{N}$ and $\mathtt{M}$ is not empty. For proving 2. we consider an instance I of Post's correspondence problem. We can reduce I to a graph $\mathcal{D}$ over a fixed alphabet and separately construct a single query Q such that $\langle\rangle \in \mathrm{Q}_{\mathcal{D}}$ if and only if instance I has a solution. □

Theorem 2 can be strengthened: query evaluation is already undecidable whenever the formalisms used to specify the labeling of paths can express languages that are the intersection of two context-free grammars. Examples of such grammar formalisms are the alternating context-free grammars [25], the conjunctive grammars [26], and many other mildly context-sensitive grammar formalisms.

Theorem 2 does however strongly depend on graphs having cycles. One can easily build recognizers for two context-free grammars and check if a finite string is accepted by each context-free grammar (and thus by the intersection of the context-free grammars). Hence, we have the following:

PROPOSITION 2. *Query evaluation of CCFPQ with path variables on directed acyclic graphs is decidable.*

The naive algorithm for the query evaluation problem of Proposition 2 would try every possible path. It is unknown if better solutions exist, as the exact complexity of query evaluation of CCFPQ with path variables on directed acyclic graphs remains to be determined.

3.3 Querying for paths

A second usage of path variables in CRPQ is specifying which paths should be returned by a query. Similarly to the situation for CRPQ [5], we show that querying for (sets of) paths is possible in the setting of context-free grammars.

Graphs can be seen as a description of traces, this in the form of a *non-deterministic finite automaton*. These automata describe regular languages. It is well-known that the intersection of a regular language (the input graph) and a context-free grammar is in itself representable by a context-free grammar. This intersection property can even be used as the basis for interesting parsing algorithms [3, 14]. We thus can use a context-free grammar representation of all the paths that matches the context-free relations used in a query.

For many practical applications, the possibility to extract a single path is much more valuable. Examples are applications for debugging queries and for analyzing graphs. Due to practical limitations, it would be preferable if we can extract a path of minimal length. For context-free grammars, finding the shortest string that can be produced starting from a non-terminal is decidable [23]. With sufficient bookkeeping, the production of the shortest string s can also be used to produce the the nodes and edges of the path π with $\mathcal{T}(\pi) = s$.

3.4 Context-free grammars over tuples

CCFPQ, as does CRPQ, lack mechanisms to compare paths. In the case of CRPQ, path comparisons have been added by generalizing regular expressions to *regular relations* [4, 5, 13]. Usually, these regular relations are specified by regular expressions over tuples of edge labels and are used in combination with path variables resulting in the *Extended Conjunctive Regular Path Queries* (ECRPQ) [5].

Based on extending CRPQ with regular relations, we extend CCFPQ with i-ary context-free relations. In view of Theorem 2, we choose an approach without using path variables. We do so by specifying paths using begin and end nodes. We remark that in the setting of ECRPQ, it can be proven that every query can be rewritten such that every path variable is used at most once [13]. Using a path variable once, is effectively equivalent to specifying begin and end nodes.

We define i-ary context-free relations similarly to the way i-ary rational relations are defined [4]. Let $\varsigma \notin \mathcal{A} \cup A$ be the *skip-symbol*. We use the skip-symbol to pad traces of paths to *skip-traces* of arbitrary lengths. The *skip-traces* of path π, with $\mathcal{T}(\pi) = l_1 \dots l_i$, are defined as

$$\mathcal{T}_\varsigma(\pi) = \{s_1 + l_1 + s_2 + \dots + s_i + l_i + s_{i+1} \mid s_1, \dots, s_{i+1} \text{ are finite sequences of } \varsigma \text{ symbols}\}.$$

The *skip-traces* of an i-tuple of paths $\vec{\pi} = \langle \pi_1, \dots, \pi_i \rangle$ are defined by

$$\mathcal{T}_{\varsigma i}(\vec{\pi}) = \{\langle t_1^1, \dots, t_1^i \rangle \dots \langle t_l^1, \dots, t_l^i \rangle \mid \forall j \; ((1 \le j \le i) \Longrightarrow ((t^j = t_1^j \dots t_l^j) \wedge (t^j \in \mathcal{T}_\varsigma(\pi_j))))\}.$$

Example 4. Let $\mathcal{D}$ be the graph on the left of Figure 3. Let path π be the path in this graph with trace $\mathcal{T}(\pi) = ac$. The skip-traces with length at most three of path π are ac, ςac, $a\varsigma c$, and $ac\varsigma$. The skip-traces of the tuple of paths $\langle \pi, \pi \rangle$ of length at most three include $\langle a, a \rangle \langle c, c \rangle$, $\langle \varsigma, \varsigma \rangle \langle a, a \rangle \langle c, c \rangle$, and $\langle a, a \rangle \langle \varsigma, c \rangle \langle c, \varsigma \rangle$.

A *i-ary context-free grammar* over alphabet A is a context-free grammar of the form $G = (N, (A \cup \varsigma)^i, P)$. Given a graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ and grammar $G = (N, (A \cup \varsigma)^i, P)$, we define *$i$-ary context-free relations* $\mathcal{R}_{\mathtt{N}} \subseteq \langle \mathcal{V} \times \mathcal{V} \rangle^i$, for every $\mathtt{N} \in N$, such that

$$\mathcal{R}_{\mathtt{N}} = \{(\langle n_1, m_1 \rangle, \dots, \langle n_i, m_i \rangle) \mid \exists n_1 \pi_1 m_1 \dots \exists n_i \pi_i m_i \; \exists t \; ((t \in \mathcal{T}_{\varsigma i}(\langle \pi_1, \dots, \pi_i \rangle)) \wedge (t \in \mathcal{L}(G_{\mathtt{N}})))\}.$$

Using this interpretation of non-terminals over graphs, we define the *Extended Conjunctive Context-Free Path Queries* as follows:

Definition 3. An *Extended Conjunctive Context-Free Path Query* (ECCFPQ) is an expressions of the form:

$$\mathrm{Q}(\vec{\nu}) \leftarrow \exists \vec{\mu} \bigwedge_{i \in \mathcal{I}} \mathtt{N}_i(\vec{p}_i),$$

where Q is the name of the query, $\vec{\nu}$ is a tuple of node variables, $\vec{\mu}$ is a tuple of distinct node variables that do not occur in $\vec{\nu}$, i ranges over a finite index set $\mathcal{I}$, $\mathtt{N}_i$ is a non-terminal from a j-ary grammar, and $\vec{p}_i$ is a j-ary tuple of pairs of node variables taken from $\vec{\nu}$ or $\vec{\mu}$.

The semantic evaluation of query Q on graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, denoted by $\mathrm{Q}_{\mathcal{D}}$, is defined by the usual semantics of first-order queries over $(\mathcal{V}; \mathcal{R}_{\mathtt{N}_i}, i \in \mathcal{I})$, whereby atoms $\mathtt{N}_i(\vec{p}_i)$ are interpreted as the relations $\mathcal{R}_{\mathtt{N}_i}(\vec{p}_i)$. We allow that each $\mathtt{N}_i$ is taken from different context-free grammars (each grammar having its own arity). We define boolean queries in the usual way.

Example 5. Consider the following binary grammar:

$$\mathtt{Sub} \to \lambda, \quad \mathtt{Sub} \to \langle \sigma, \sigma \rangle \, \mathtt{Sub} \; (\forall \sigma \in A), \quad \mathtt{Sub} \to \langle \sigma, \varsigma \rangle \, \mathtt{Sub} \; (\forall \sigma \in A).$$

The query $\mathrm{Q}_{\mathtt{Sub}}(n_1, m_1, n_2, m_2) \leftarrow \mathtt{Sub}(\langle n_1, m_1 \rangle, \langle n_2, m_2 \rangle)$ will return node-pairs (n_1, m_1) and (n_2, m_2) such that there are paths $n_1 \pi_1 m_1$ and $n_2 \pi_2 m_2$ whereby $\mathcal{T}(\pi_2)$ is a *subsequence* of $\mathcal{T}(\pi_1)$. A string s_2 is a subsequence of s_1 if s_2 can be obtained by removing symbols from s_1. Figure 2 illustrates subsequences in a graph with two paths connecting node pairs $\langle n_1, m_1 \rangle$ and $\langle n_2, m_2 \rangle$.

In the graph of Figure 2, the path $n_1 \pi_1 m_1$ has trace $\mathcal{T}(\pi_1) = eaceca$. When we remove the first occurrence of e, the first occurrence of c, and the last of occurrence of a, then the

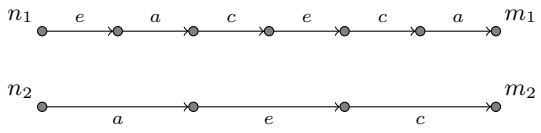

Figure 2: Two node pairs $\langle n_1, m_1\rangle$ and $\langle n_2, m_2\rangle$ of begin and end nodes connected by different paths. The trace of the shorter path is a subsequence of the trace of the longer path.

result is *aec*. This is equivalent to the trace $\mathcal{T}(\pi_2)$ of the path $n_2\pi_2 m_2$. Hence, we conclude that *aec* is a subsequence of *eaceca* and that $(\langle n_1, m_1\rangle, \langle n_2, m_2\rangle) \in \mathcal{R}_{\texttt{Sub}}$.

3.4.1 Query evaluation

We reduce query evaluation of ECCFPQ to query evaluation of CCFPQ by calculating the relations $\mathcal{R}_{\texttt{N}}$, for N a non-terminal from an i-ary context-free grammar, using Algorithm 1. We do so in a similar way as the reduction of query evaluation of ECRPQ to CRPQ [5].

The *i-product graph* of graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, denoted by $\mathcal{D}_{\times i} = (\mathcal{V}_{\times i}, \mathcal{E}_{\times i})$, is defined by $\mathcal{V}_{\times i} = \{\langle n_1, \ldots, n_i\rangle \mid n_1, \ldots, n_i \in \mathcal{V}\}$ and

$$\mathcal{E}_{\times i} = \{(n^i, \langle l_1, \ldots, l_i\rangle, m^i) \mid (n^i, m^i \in \mathcal{V}_{\times i}) \wedge \forall j\ ((1 \leq j \leq i) \implies ((n^i_j, l_j, m^i_j) \in \mathcal{E} \vee (n^i_j = m^i_j \wedge l_j = \varsigma)))\}.$$

Proposition 3. *Let $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ be a graph. We can construct the i-product graph $\mathcal{D}_{\times i} = (\mathcal{V}_{\times i}, \mathcal{E}_{\times i})$ in $O((|\mathcal{V}| + |\mathcal{E}|)^i)$.*

Now, by construction, every path $n^i \pi m^i$ in $\mathcal{D}_{\times i}$ is a path whose trace simulates an i-skip-trace between nodes $\langle n^i_1, m^i_1\rangle$, $\ldots$, $\langle n^i_i, m^i_i\rangle$. Hence, we can use Algorithm 1 and graph $\mathcal{D}_{\times i}$ to calculate $\mathcal{R}_{\texttt{N}}$.

Example 6. Let $\mathcal{D}$ be the graph on the left of Figure 3. The 2-product graph of this graph is displayed on the right of Figure 3.

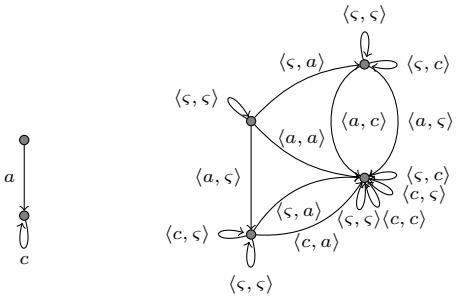

Figure 3: A graph on the left and its 2-product graph on the right.

The pair of traces $\langle ac, acc\rangle$ in the graph on the left of Figure 3 is represented by an infinite number of pairs of traces in the graph on the right of Figure 3. Examples include the traces $\langle a, a\rangle\langle\varsigma, c\rangle\langle c, c\rangle$ and $\langle a,\varsigma\rangle\langle\varsigma, a\rangle\langle\varsigma, c\rangle\langle c,\varsigma\rangle\langle\varsigma, c\rangle$. These traces are indeed examples of skip-traces of a 2-tuple of paths $\langle\pi_1, \pi_2\rangle$ with $\mathcal{T}(\pi_1) = ac$ and $\mathcal{T}(\pi_2) = acc$.

Corollary 2. *Let $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ be a graph and let $G = (N, A, P)$ be an i-ary grammar. Algorithm 1 applied to $\mathcal{D}$ and G has a worst-case complexity of $\mathcal{O}(|N|(|\mathcal{V}| + |\mathcal{E}|)^i + (|N||\mathcal{V}|^i)^3)$.*

As a direct consequence of Corollary 1 and Corollary 2, we have the following:

Corollary 3. *For ECCFPQ queries, the query evaluation problem has*

1. *Polynomial time data complexity,*
2. *NP-complete combined complexity.*

We remark that the known combined complexity of ECRPQ is PSPACE-complete [5]. This is not in contradiction with Corollary 3, as ECRPQ traditionally uses path variables. Path variables allow a more succinct notation of queries involving, for example, intersections of regular languages.

3.4.2 Regular and rational relations

In the setting of CRPQ, there is a distinction between *regular relations* and *rational relations* [4, 13]. The difference being that regular relations specify skip-traces that are only padded from the right. Hence, in regular relations the skip-traces of path π, with $\mathcal{T}(\pi) = l_1 \ldots l_i$, are defined as

$$\mathcal{T}_\varsigma(\pi) = \{l_1 + \ldots + l_i + s \mid s \text{ is a finite sequence of } \varsigma \text{ symbols}\}.$$

The rational relations allow arbitrary padding, as in i-ary context-free grammars. In the presence of path variables, the distinction between regular and rational relations is important. Similar to Theorem 2, we have the following.

Theorem 3 (Barceló, Figueira, Libkin [4]). *The query evaluation problem of CRPQ using path variables extended with rational relations is undecidable, even for a fixed query.*

Instead of considering CRPQ with rational relations in the presence of path variables, we can consider CRPQ with rational relations using begin and end nodes to specify paths. Effectively, such a variant on CRPQ simply restricts the context-free grammars used in ECCFPQ to regular grammars. We say that an ECCFPQ built over a regular grammar is an *Extended Conjunctive Regular Path Query with skips* (ECRPQ$^\varsigma$).

For completeness, we can also define the context-free generalization of regular relations, as used in ECRPQ. We do

so in the *Extended Conjunctive Context-Free Path Queries without skips* (ECCFPQ$^{\perp}$) by restricting skip-traces to skip-traces that are only padded from the right.

PROPOSITION 4. *Every ECCFPQ$^{\perp}$ query is expressible in ECCFPQ, every ECRPQ query is expressible in ECRPQ$^{\text{s}}$.*

PROOF. The acceptable skip-traces of tuples of paths allowed by the definition of ECCFPQ$^{\perp}$ (ECRPQ) are expressible by a regular expression r. The intersection of a regular expression r and a context-free grammar (regular grammar) is itself context-free (regular). With the resulting intersections, we can express the semantics of ECCFPQ$^{\perp}$ (ECRPQ) in ECCFPQ (ECRPQ$^{\text{s}}$). □

In the following section, we further study the differences in the expressive power of CRPQ, CCFPQ, and these generalizations.

4. RELATIONS WITH CRPQ AND ECRPQ

We introduced CCFPQ and ECCFPQ as the context-free versions of respectively CRPQ and ECRPQ. We will now investigate if these extensions actually add expressive power. We use the following notations for comparing expressive powers of query languages.

Definition 4. Let L_1 and L_2 be query languages. Query language L_1 is *as expressive as* L_2, denoted by $L_2 \preceq L_1$, when every query expressible in L_2 is expressible in L_1. Query language L_1 is *strictly more expressive than* L_2, denoted by $L_2 \prec L_1$, when $L_2 \preceq L_1$ and $L_1 \not\preceq L_2$. Query language L_1 is *equal to* L_2, denoted by $L_2 \cong L_1$, when $L_1 \preceq L_2$ and $L_2 \preceq L_1$.

4.1 Relations on strings

The various graph query languages we study only differ in their usage of formal languages to specify traces. A logical step would thus be to define how the graph query languages can be used to define formal languages and then to compare the classes of formal languages definable by graph query languages. We define the language of a graph query language in terms of *string-graphs*.

Definition 5. Let $s = s_1 \dots s_j$ be a string over alphabet A with $\bullet \notin A$. The *string-graph* of s is defined as the chain of edges

$$(n_0, \bullet, n_1), (n_1, s_1, n_2), \dots, (n_j, s_j, n_{j+1}), (n_{j+1}, \bullet, n_{j+2}).$$

In string-graphs, the symbol $\bullet$ is used to bridge the semantic gap between, on the one hand, formal grammars, whose semantics is applied on entire strings, and, on the other hand, CRPQ and CCFPQ, whose existential semantics looks at any possible substring.

Example 7. The regular expression $(aa)^*$ defines the language $\mathcal{L}((aa)^*) = \{(aa)^n \mid 0 \leq n\}$. This language is not monotone: when $s \in \mathcal{L}((aa)^*)$, then $s +\!\!+ a \notin \mathcal{L}((aa)^*)$. CRPQ and CCFPQ are, however, monotone with respect to graphs. This apparent paradox is resolved by string-graphs. Consider the string $s = s_1 \dots s_i$, the string-graph of s consists of a path π with $\mathcal{T}(\pi) = \bullet s_1 \dots s_i \bullet$ and the string-graph of $s +\!\!+ a$ consists of a path π' with $\mathcal{T}(\pi') = \bullet s_1 \dots s_i a \bullet$. The path π is not contained in path π'. Hence, the modification from π to π' is non-monotone.

In the setting of string-graphs, we define the languages accepted by boolean graph query languages.

Definition 6. Let Q be a boolean graph query. The *language* of Q, denoted by $\mathcal{L}(\text{Q})$, is defined by

$$\mathcal{L}(\text{Q}) = \{s \mid \mathcal{D}_s \text{ is the string-graph of string } s \wedge \text{Q}_{\mathcal{D}_s} \neq \emptyset\}.$$

Example 8. The language $\mathcal{L}_{abc} = \{a^n b^n c^n \mid 0 \leq n\}$ is not context-free, but is definable by CCFPQ. Consider the following parameterized context-free grammar.

$$\texttt{Path}_\gamma \to \lambda, \quad \texttt{Path}_\gamma \to \gamma\, \texttt{Path}_\gamma,$$
$$\texttt{Paren}_{\alpha,\beta} \to \lambda, \quad \texttt{Paren}_{\alpha,\beta} \to \alpha\, \texttt{Paren}_{\alpha,\beta}\, \beta,$$
$$\texttt{AB} \to \bullet\, \texttt{Paren}_{a,b}\, \texttt{Path}_c\, \bullet, \quad \texttt{BC} \to \bullet\, \texttt{Path}_a\, \texttt{Paren}_{b,c}\, \bullet\,.$$

Over this grammar we construct the query

$$\text{Q}_{a^n b^n c^n}() \leftarrow \exists m_1 m_2\ \texttt{AB}(m_1, m_2) \wedge \texttt{BC}(m_1, m_2).$$

We have $\mathcal{L}(\text{Q}_{a^n b^n c^n}) = \mathcal{L}_{abc}$. The language $\mathcal{L}_{ww} = \{w +\!\!+ \circ +\!\!+ w \mid w \text{ is a string over } A \setminus \{\circ\}\}$ is not context-free, but is definable by ECRPQ. Consider the following regular grammar.

$$\texttt{Equal} \to \lambda, \quad \texttt{Equal} \to \langle \sigma, \sigma \rangle\, \texttt{Equal}\ (\forall \sigma \in A \setminus \{\circ\}),$$
$$\texttt{Mirror} \to \langle \bullet, \circ \rangle\, \texttt{Equal}\, \langle \circ, \bullet \rangle.$$

Over this grammar we construct the query

$$\text{Q}_{ww}() \leftarrow \exists n_1 n_2 m_1 m_2\ \texttt{Mirror}(\langle n_1, m_1 \rangle, \langle n_2, m_2 \rangle).$$

We have $\mathcal{L}(\text{Q}_{ww}) = \mathcal{L}_{ww}$.

Example 8 shows that CCFPQ and ECRPQ can already define languages that are not definable by context-free grammars. This is not the case for CRPQ, Freydenberger and Schweikardt actually showed the following:

THEOREM 4 (FREYDENBERGER, SCHWEIKARDT [13]). *Let $\mathcal{L}$ be a language. There is a CRPQ query* Q *with $\mathcal{L}(\text{Q}) = \mathcal{L}$ if and only if there is a regular expression r with $\mathcal{L}(r) = \mathcal{L}$.*

Theorem 4 also follows from the results in Barceló et al. [5]. By Theorem 4, CRPQ and regular grammars can define the same languages. Furthermore, as the regular grammars are strictly subsumed by the context-free grammars, we have the following:

COROLLARY 4. *On string-graphs we have*

$$\textit{CRPQ} \prec \textit{CCFPQ and CRPQ} \prec \textit{ECRPQ}.$$

Example 8 showed that CCFPQ can define languages that are not definable by context-free grammars, we strengthen these results. By $\mathfrak{L}^Q$, with $Q \in \{\text{CCFPQ}, \text{ECCFPQ}^\perp\}$, we denote the class of all languages expressible by Q queries. By $\mathfrak{L}^k$ we denote the class of all languages that are the intersection of at most k context-free languages, and we define $\mathfrak{L}^\infty = \bigcup_k \mathfrak{L}^k$. For all $1 \leq k$, we have $\mathfrak{L}^k \subset \mathfrak{L}^{k+1}$, and we have $\mathcal{L}_{ww} \notin \mathfrak{L}^\infty$ [21, 34].

PROPOSITION 5. *For every $\mathcal{L} \in \mathfrak{L}^k$, there is a CCFPQ query* Q *with at most k relation atoms such that $\mathcal{L}(\text{Q}) = \mathcal{L}$.*

COROLLARY 5. *We have $\mathfrak{L}^\infty \subseteq \mathfrak{L}^{CCFPQ}$, and we have $\mathfrak{L}^\infty \subset \mathfrak{L}^{ECCFPQ^\perp}$.*

4.2 Relations on unlabeled graphs

It is well-known that on strings over a unary alphabet the context-free grammars and regular grammars have equivalent expressive power. This equivalence follows directly from Parikh's theorem [27]. We investigate if there is a similar equivalence for the graph query languages on unlabeled graphs. We represent unlabeled graphs as graphs where every edge is labeled with the edge label u.

Definition 7. We define the *length* of u as $\ell(u) = 1$ and the *length* of ς as $\ell(\varsigma) = 0$. The *length-tuple* of a tuple of edge labels $\vec{t} = \langle t_1, \ldots, t_i \rangle$ is defined by the vector of natural numbers $\ell(\vec{t}) = \langle \ell(t_1), \ldots, \ell(t_i) \rangle$. The *length-tuple* of string $s = s_1 \ldots s_j$ over the alphabet $\{u, \varsigma\}^i$ is defined as $\ell(s) = \ell(s_1) + \cdots + \ell(s_j)$ (using vector addition).

Let $G = (N, \{u, \varsigma\}^i, P)$ be a i-ary grammar. The *length-language* with respect to non-terminal $\mathtt{N} \in N$ is defined by $\mathcal{L}_\ell(G_\mathtt{N}) = \{\ell(s) \mid s \in \mathcal{L}(G_\mathtt{N})\}$.

Example 9. Consider the following grammar:

$$\mathtt{N} \to \langle u, u \rangle \langle u, \varsigma \rangle \mathtt{N}, \quad \mathtt{N} \to \langle u, \varsigma \rangle.$$

We have $\mathcal{L}(G_\mathtt{N}) = \{(\langle u, u \rangle \langle u, \varsigma \rangle)^n \langle u, \varsigma \rangle \mid 0 \leq n\}$. We thus have $\mathcal{L}_\ell(G_\mathtt{N}) = \{n \cdot (\langle 1, 1 \rangle + \langle 1, 0 \rangle) + \langle 1, 0 \rangle \mid 0 \leq n\}$.

Intuitively, queries over $\{u, \varsigma\}^i$ accept tuples of paths based only on the length of the individual paths. We prove this intuition to be correct.

PROPOSITION 6. *Let G^N and G^M be grammars over alphabet $\{u, \varsigma\}^i$ and let $\mathtt{N}$ and $\mathtt{M}$ be non-terminals such that $\mathcal{L}_\ell(G^N{}_N) = \mathcal{L}_\ell(G^M{}_M)$. We have $(\langle n_1, m_1 \rangle, \ldots, \langle n_i, m_i \rangle) \in \mathcal{R}_N$ if and only if $(\langle n_1, m_1 \rangle, \ldots, \langle n_i, m_i \rangle) \in \mathcal{R}_M$.*

PROOF. We only prove the if direction. The proof for the only if direction is analogous by interchanging the role of $G^\mathtt{N}$ and $G^\mathtt{M}$. If $(\langle n_1, m_1 \rangle, \ldots, \langle n_i, m_i \rangle) \in \mathcal{R}_\mathtt{N}$, then, for all $1 \leq j \leq i$, there exists paths $n_j \pi_j m_j$ such that there is an $s \in \mathcal{T}_{\varsigma i}(\pi_1, \ldots, \pi_i)$ with $s \in \mathcal{L}(G^\mathtt{N}{}_\mathtt{N})$. We have $\mathcal{L}_\ell(G^\mathtt{N}{}_\mathtt{N}) = \mathcal{L}_\ell(G^\mathtt{M}{}_\mathtt{M})$. Hence, there is an $s' \in \mathcal{L}(G^\mathtt{M}{}_\mathtt{M})$ with $\ell(s) = \ell(s')$. By the definition of $\mathcal{T}_{\varsigma i}$, we have $s' \in \mathcal{T}_{\varsigma i}(\pi_1, \ldots, \pi_i)$ and thus $(\langle n_1, m_1 \rangle, \ldots, \langle n_i, m_i \rangle) \in \mathcal{R}_\mathtt{M}$. □

Using the above result we prove the following result.

THEOREM 5. *Let $\mathtt{N}$ be a non-terminal in an i-ary context-free grammar G over $\{u, \varsigma\}^i$. There exists a non-terminal $\mathtt{M}$ in a i-ary regular grammar such that $\mathcal{R}_N = \mathcal{R}_M$.*

PROOF. Arbitrary permutations of strings over $\{u, \varsigma\}^i$ do not affect the length-language of the strings. Hence, Parikh's theorem teaches us that the length-language of $G_\mathtt{N}$ is definable by a regular grammar. Let $\mathtt{M}$ be the start non-terminal in such a i-ary regular grammar G' defining the same length-language as $G_\mathtt{N}$. By construction we have $\mathcal{L}_\ell(G_\mathtt{N}) = \mathcal{L}_\ell(G'_\mathtt{M})$, hence we can apply Proposition 6. □

Remark that Theorem 5 reduces i-ary context-free grammars, as used in ECCFPQ$^\perp$, to rational relations, as used in ECRPQ$^\varsigma$, and not to regular relations, as used in ECRPQ.

COROLLARY 6. *On unlabeled graphs we have*

$$\textit{CRPQ} \cong \textit{CCFPQ and ECRPQ}^\varsigma \cong \textit{ECCFPQ}.$$

It has already been proven that CRPQ $\prec$ ECRPQ for non-boolean queries on unlabeled graphs [13]. Hence, we also have the following:

COROLLARY 7. *On unlabeled graphs we have*

$$\textit{CCFPQ} \prec \textit{ECRPQ and CCFPQ} \prec \textit{ECCFPQ}^\perp.$$

4.3 Overview

Figure 4 provides the implication of the results on strings and on unlabeled graphs for the general relations of the expressive power of CRPQ, CCFPQ, and their generalizations on graphs.

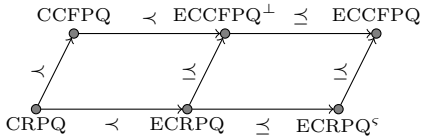

Figure 4: Known relations between variants of CRPQ and CCFPQ.

5. INHERENT LIMITS OF PATH QUERIES BASED ON FORMAL LANGUAGES

The context-free relations are defined independent of the conjunctive query framework of CCFPQ. We can also consider using context-free grammars in other logics and query languages. A logical extension of CCFPQ is the addition of negation. This is in line with similar additions to CRPQ [5]. We add negation in two flavors: adding negated atoms, and adding negation as a general boolean operator. We then investigate the differences between the expressive power of the resulting query languages.

The differences in the expressive power of the resulting query languages are largely independent of the class of formal languages $\mathfrak{L}$, with language $\mathcal{L} \in \mathfrak{L}$, used to define i-ary relations

$\mathcal{R}_{\mathcal{L}}$. We thus present our results for all *Conjunctive* $\mathfrak{L}$*-Path Queries* (CLPQ), where $\mathfrak{L}$ is a class of formal languages used to define the relational atoms used in CLPQ. This in the same way as context-free grammars and i-ary context-free grammars are used to define relations atoms in CCFPQ and ECCFPQ, respectively. To the query language CLPQ we add negated atoms, resulting in CLPQ$^{\neg}$, and we add negation as a general boolean operator, resulting in CLPQ$^{\text{FO}}$.

To keep expressivity results as general as possible, we restrict ourselves to regular languages in examples of queries expressible in CLPQ, CLPQ$^{\neg}$, and CLPQ$^{\text{FO}}$.

5.1 Results for CLPQ

The query language CLPQ is monotone and closed under homomorphism. This places restrictions on the expressive power of CLPQ that are easy to check.

PROPOSITION 7. *The boolean queries* Is graph $\mathcal{D}$ connected, acyclic, bipartite, Hamiltonian, or Eulerian?[2] *and the boolean queries* Does graph $\mathcal{D}$ have/not have source/sink nodes?[3]*, and* Does there exists a node with two outgoing edges? *are not expressible in CLPQ.*

PROOF. Only the last query is monotone. For this last query, we use a homomorphism argument.

Let $\mathcal{D}_1$ be the graph on the left of Figure 5 and let $\mathcal{D}_2$ be the graph on the right of Figure 5. We provide a homomorphism $h_{12} : \mathcal{V}_1 \to \mathcal{V}_2$ from $\mathcal{D}_1$ to $\mathcal{D}_2$ and a homomorphism $h_{21} : \mathcal{V}_2 \to \mathcal{V}_1$ from $\mathcal{D}_2$ to $\mathcal{D}_1$ by mapping roots to roots and leafs to leafs.

Figure 5: Graphs that cannot be distinguished by boolean CLPQ.

By homomorphisms h_{12} and h_{21}, CLPQ cannot distinguish the graphs in Figure 5. □

5.2 CLPQ with negated atoms (CLPQ$^{\neg}$)

In the context of CCFPQ, we add negated atoms by not only allowing atoms of the form $\mathtt{N}(n, m)$, but also allow negated atoms of the form $\neg\mathtt{N}(n, m)$. Equivalently, we add the complements $\overline{\mathcal{R}_{\mathtt{N}}}$ of context-free relations $\mathcal{R}_{\mathtt{N}}$ to the query language. We remark that these complement relations can be constructed alongside the original relations using Algorithm 1, this without affecting complexity.

For clarity, we remark that $\overline{\mathcal{R}_{\mathcal{L}}}$ and $\mathcal{R}_{\overline{\mathcal{L}}}$ are different relations, this is easily verified using Example 2: $(m, n) \in \overline{\mathcal{R}_{\mathtt{N}}}$ and $(m, n) \notin \mathcal{R}_{\overline{\mathtt{N}}}$. Straightforwardly, CLPQ$^{\neg}$ is not monotone and not closed under homomorphism.

Example 10. The boolean query *Is graph $\mathcal{D}$ not connected?* is not expressible in CLPQ, the query is however expressible in CLPQ$^{\neg}$. This by the grammar obtained by adding the production rule $\mathtt{TC} \to \lambda$ to the grammar of Example 3:

$$\mathrm{Q}_{not\text{-}connected}() \leftarrow \exists nm\ \neg\mathtt{TC}(n, m).$$

Also the boolean query *Does there exists a node with two outgoing edges?* is expressible by using the following grammar:

$$\mathtt{Edge} \to \sigma\ (\forall \sigma \in A),\ \mathtt{Equal} \to \lambda.$$

Using this grammar, we express *Does there exists a node with two outgoing edges?* by:

$$\mathrm{Q}_{branch}() \leftarrow \exists nm_1m_2\ \mathtt{Edge}(n, m_1) \wedge \mathtt{Edge}(n, m_2) \wedge \neg\mathtt{Equal}(m_1, m_2).$$

The boolean queries *Is graph $\mathcal{D}$ acyclic, bipartite, Hamiltonian, or Eulerian?* and *Does graph $\mathcal{D}$ have/not have source/sink nodes?* are inexpressible in CLPQ$^{\neg}$. Before we prove these results, we prove that CLPQ$^{\neg}$ is closed under *subgraph-mappings.*

Definition 8. Let $\mathcal{D}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{D}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be graphs. A mapping $h : \mathcal{V}_1 \to \mathcal{V}_2$ is called a *subgraph-mapping* from $\mathcal{D}_1$ to $\mathcal{D}_2$ if h is a homomorphism and if, for every pair of nodes $n, m \in \mathcal{V}_1$, we have $\mathcal{T}_{\mathcal{S}}(n, m) = \mathcal{T}_{\mathcal{S}}(h(n), h(m))$.

A query language L is *closed under subgraph-mappings* if, for every i-ary query Q expressible in L, for every pair of graphs $\mathcal{D}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{D}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, and for every subgraph-mapping $h : \mathcal{V}_1 \to \mathcal{V}_2$ from $\mathcal{D}_1$ to $\mathcal{D}_2$, we have $\langle n_1, \ldots, n_i \rangle \in \mathrm{Q}_{\mathcal{D}_1}$ implies $\langle h(n_1), \ldots, h(n_i) \rangle \in \mathrm{Q}_{\mathcal{D}_2}$.

Intuitively, subgraph-mappings map the nodes of one graph into the nodes of another graph, while maintaining the set of traces between pairs of mapped nodes. Homomorphisms map nodes of one graph into the nodes of another graph, while only maintaining a superset of the set of traces between pairs of mapped nodes.

THEOREM 6. *CLPQ and CLPQ$^{\neg}$ are closed under subgraph-mappings.*

We are now ready to prove the inexpressibility claims made after Example 10.

PROPOSITION 8. *The boolean queries* Is graph $\mathcal{D}$ acyclic, bipartite, Hamiltonian, or Eulerian? *and* Does graph $\mathcal{D}$ have/not have source/sink nodes? *are not expressible in CLPQ$^{\neg}$.*

[2]A graph is *connected* if and only if there is a directed path between every pair of nodes. A graph is *acyclic* if and only if it does not contain a cycle. A graph is *bipartite* if and only if the set of nodes $\mathcal{V}$ of the graph can be partitioned in two sets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that, for every edge (n, σ, m), we have $n \in \mathcal{V}_1, m \in \mathcal{V}_2$ or $n \in \mathcal{V}_2, m \in \mathcal{V}_1$. A graph is *Hamiltonian* if and only if there is a directed path containing every node exactly once. A graph is *Eulerian* if and only if there is a directed path containing every edge exactly once.

[3]In graphs, the *source nodes* are the nodes that do not have incoming edges, an example of a source node is the root node of a tree. The *sink nodes* are the nodes that do not have outgoing edges, examples of sink nodes are the leafs of a tree.

PROOF. Let $\mathcal{D}_1$ be an acyclic graph and let Q be a CLPQ$^\neg$ query with $\langle\rangle \in \mathrm{Q}_{\mathcal{D}_1}$. By Theorem 6, we have $\langle\rangle \in \mathrm{Q}_{\mathcal{D}_2}$, where $\mathcal{D}_2$ is constructed from $\mathcal{D}_1$ by adding a separate cyclic subgraph. For the queries *Does graph $\mathcal{D}$ not have source/sink nodes?*, we take a similar approach.

For the queries *Does graph $\mathcal{D}$ have source/sink nodes?*, we consider the graphs in Figure 6.

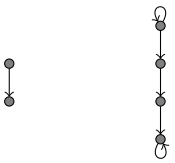

Figure 6: Graphs that cannot be distinguished by boolean CLPQ$^\neg$.

Only the graph on the left has source/sink nodes, and there is a subgraph mapping to the graph on the right.

In Section 5.3, we will prove that CLPQ$^{\text{FO}}$, a generalization of CLPQ$^\neg$, cannot express the queries *Is graph $\mathcal{D}$ Hamiltonian or Eulerian?*. □

5.3 CLPQ with negation (CLPQ$^{\text{FO}}$)

The result of adding negation to CLPQ, is the first-order logic over $(\mathcal{V}; \mathcal{R}_{\mathcal{L}_i}, i \in \mathcal{I})$. To simplify notations, we can also include the short-hand notations $\vee$ and $\forall$ with their usual meaning. In the presence of path variables, the addition of negation as a general boolean operator does affect the complexity of query evaluation: the combined complexity of CRPQ$^{\text{FO}}$ is PSPACE-complete and the data and combined complexity of ECRPQ$^{\text{FO}}$ is non-elementary [5].

Example 11. The boolean queries *Is graph $\mathcal{D}$ acyclic?* and *Does graph $\mathcal{D}$ have/not have source/sink nodes?* are not expressible in CLPQ$^\neg$. These queries are however expressible in CLPQ$^{\text{FO}}$. We express *Is graph $\mathcal{D}$ acyclic?* using the grammar of Example 3:

$$\mathrm{Q}_{acyclic}() \leftarrow \neg\exists n\ \mathtt{TC}(n, n).$$

We express *Does graph $\mathcal{D}$ have source nodes?* using the grammar of Example 10:

$$\mathrm{Q}_{has\text{-}source}() \leftarrow \exists n \forall m\ \neg\mathtt{Edge}(m, n).$$

The other three source/sink queries are expressible in a similar manner. Also the boolean query *Is graph $\mathcal{D}$ bipartite?* is expressible in CLPQ$^{\text{FO}}$. A graph $\mathcal{D}$ is bipartite if and only if $\mathcal{D}$ does not contain an odd cycle [9]. We use the following grammar:

$$\mathtt{Odd} \rightarrow \lambda, \quad \mathtt{Even} \rightarrow \mathtt{Odd}\,\sigma\ (\forall \sigma \in A),$$
$$\mathtt{Odd} \rightarrow \mathtt{Even}\,\sigma\ (\forall \sigma \in A).$$

Using this grammar, we can express *Is graph $\mathcal{D}$ bipartite?* by:

$$\mathrm{Q}_{bipartite}() \leftarrow \forall n\ \neg\exists m\ \mathtt{Odd}(n, m) \wedge \mathtt{Edge}(m, n).$$

The boolean queries *Is graph $\mathcal{D}$ Hamiltonian or Eulerian?* remain inexpressible in CLPQ$^{\text{FO}}$. For first-order logic, we have well-established and complete methods to prove inexpressibility results in the form of Ehrenfeucht-Fraïssé games [19]. These methods can also be used to prove inexpressibility results for CLPQ$^{\text{FO}}$.

PROPOSITION 9. *Let $\mathcal{D}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{D}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be graphs. When the Duplicator has a winning strategy for Ehrenfeucht-Fraïssé games on the structures $(\mathcal{V}_1; \mathcal{R}_{\mathcal{L}_i}, i \in \mathcal{I})$ and $(\mathcal{V}_2; \mathcal{R}_{\mathcal{L}_i}, i \in \mathcal{I})$, then no CLPQ$^{\text{FO}}$ query can distinguish $\mathcal{D}_1$ from $\mathcal{D}_2$.*

We provide a sufficient condition for such a winning strategy based on a generalization of the strategy used to prove that *Regular Walk Logic* cannot express *Is graph $\mathcal{D}$ Hamiltonian?* [15].

Definition 9. Let $\mathcal{D}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{D}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be graphs. Let $a_1, \ldots, a_r \in \mathcal{V}_1$ and $b_1, \ldots, b_r \in \mathcal{V}_2$ be the moves in an r-round Ehrenfeucht-Fraïssé game. These moves are a *trace-set winning position* for the Duplicator if we have $\mathcal{T}_{\mathcal{S}}(a_i, a_j) = \mathcal{T}_{\mathcal{S}}(b_i, b_j)$ for all i and j with $1 \leq i \leq r$ and $1 \leq j \leq r$.

The Duplicator has a *r-round trace-set winning strategy* if the Duplicator can play in a way that guarantees a trace-set winning position after r rounds.

THEOREM 7. *Let $\mathcal{D}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{D}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be graphs. When the Duplicator has a r-round trace-set winning strategy for Ehrenfeucht-Fraïssé games on $\mathcal{D}_1$ and $\mathcal{D}_2$, then the Duplicator also has a r-round winning strategy for Ehrenfeucht-Fraïssé games on structures $(\mathcal{V}_1; \mathcal{R}_{\mathcal{L}_i}, i \in \mathcal{I})$ and $(\mathcal{V}_2; \mathcal{R}_{\mathcal{L}_i}, i \in \mathcal{I})$.*

PROOF. Let $\mathcal{L}$ be an arbitrary language over $(A \cup \varsigma)^n$. We need to prove that a r-round trace-set winning strategy implies $(\langle a_{i_1}, a_{j_1}\rangle, \ldots, \langle a_{i_n}, a_{j_n}\rangle) \in \mathcal{R}_{\mathcal{L}}$ if and only if $(\langle b_{i_1}, b_{j_1}\rangle, \ldots, \langle b_{i_n}, b_{j_n}\rangle) \in \mathcal{R}_{\mathcal{L}}$, for any $1 \leq i_k \leq r$, $1 \leq j_k \leq r$, and $1 \leq k \leq n$, this follows directly from $\mathcal{T}_{\mathcal{S}}(a_{i_k}, a_{j_k}) = \mathcal{T}_{\mathcal{S}}(b_{i_k}, b_{j_k})$. □

We are now ready to prove the inexpressibility claims made after Example 11.

PROPOSITION 10. *CLPQ$^{\text{FO}}$ cannot express the boolean queries* Is graph $\mathcal{D}$ Hamiltonian or Eulerian?*.*

PROOF (SKETCH). The proof strategies for proving inexpressibility of the boolean queries *Is graph $\mathcal{D}$ Hamiltonian or Eulerian?* can be derived from similar proofs in other work [8, 15, 17]. □

6. CONCLUSION

We have proposed the Conjunctive Context-Free Path Queries as a natural extension of the well-known Conjunctive

Regular Path Queries. We also looked at the generalization of regular expressions, as used in CRPQ, to regular relations and we showed how similar generalizations can be applied to context-free grammars, as used in CCFPQ.

In our work, we encountered an important restriction in the usage of path variables: in CCFPQ, using path variables leads to undecidability of query evaluation. The literature shows similar undecidability results when CRPQ, using path variables, is extended with rational relations [5]. We thus propose using begin and end nodes as an alternative to path variables. In this way, we were able to generalize CRPQ with rational relations and CCFPQ with context-free grammars over tuples of edge-labels.

The main results are that query evaluation of CCFPQ and ECCFPQ maintain the polynomial time data complexity and NP-complete combined complexity of CRPQ. At the same time, CCFPQ is strictly more expressive than CRPQ, even on chains, and ECCFPQ is strictly more expressive than CCFPQ, even on unlabeled graphs.

Finally, we looked at using context-free grammars in generalizations of the conjunctive queries by adding negation in two flavors: adding negated atoms and adding negation as a general boolean operator. The differences of the expressive power of these languages is largely unaffected by the particular choice of grammar formalism used. Hence, we proved a strict hierarchy between query languages without negation, with negated atoms, and with negation as a general boolean operator, this for any conjunctive query language that uses formal languages to specify the labeling of paths.

Given these results, we identify a number of open problems for further research. First, we did not yet fully explore the relations in the expressive power of the query languages CRPQ, CCFPQ, and their generalizations. The open cases involve ECRPQ, ECRPQ$^{\varsigma}$, ECCFPQ$^{\perp}$ and ECCFPQ. These query languages embed regular relations, rational relations, and i-ary context-free grammars within a conjunctive query language. Hence, existing results on the relations between regular, rational, and context-free definable functions (e.g. [6, 7]) are not directly applicable. These existing results do however provide a starting point for future research. Secondly, we did not consider CCFPQ with simple path semantics, even though there has been some interest in the expressive power of CRPQ with simple path semantics [15], this due to the simple path semantics of property paths in SPARQL [2, 22].

Besides these open theoretical problems, there are also a number of practical open problems. It is yet unknown if our context-free recognizer for graphs is optimal. The context-free recognizer for graphs, presented in Algorithm 1, only provides a worst-case upper bound for the complexity of context-free recognizers for graphs. Due to the size of the output, we also have a theoretical lower bound of $\Omega(|\mathcal{V}|^2)$. Practically speaking, graph recognition is at least as costly as normal context-free string recognition and as costly as computing the transitive closure. The best known algorithms for solving these problems are based on boolean matrix multiplication [31], for which the currently-best algorithm has a complexity of $\mathcal{O}(n^{2.3727})$ [33], with $n \times n$ the dimensions of the matrix. Furthermore, for path querying, whereby the result of a query is a path instead of a tuple of nodes, we merely proven decidability. Path querying algorithms and the analysis of the complexity of path querying as a whole, remain interesting topics for further research.

Acknowledgment

The author thanks Jan Van den Bussche and the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, S. Conca, and J. Pérez. Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 629–638, 2012.

[3] Y. Bar-Hillel, M. A. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961.

[4] P. Barceló, D. Figueira, and L. Libkin. Graph logics with rational relations and the generalized intersection problem. In *27th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 115–124, 2012.

[5] P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37(4):31:1–31:46, 2012.

[6] J. Berstel. *Transductions and Context-Free Languages*. Leitfäden der angewandten Mathematik und Mechanik. B.G Teubner, 1979.

[7] C. Choffrut and K. Culik II. Properties of finite and pushdown transducers. *SIAM Journal on Computing*, 12(2):300–315, 1983.

[8] M. De Rougemont. Second-order and inductive definability on finite structures. *Mathematical Logic Quarterly*, 33(1):47–63, 1987.

[9] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer Heidelberg, 2010.

[10] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[11] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[12] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pages 139–148, 1998.

[13] D. D. Freydenberger and N. Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *Journal of Computer and System Sciences*, 79(6):892–909, 2013.

[14] D. Grune and C. J. H. Jacobs. *Parsing Techniques*. Monographs in Computer Science. Springer New York, 2008.

[15] J. Hellings, B. Kuijpers, J. Van den Bussche, and X. Zhang. Walk logic as a framework for path query

languages on graph databases. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 117–128. ACM, 2013.

[16] T. Kasami and K. Torii. A syntax-analysis procedure for unambiguous context-free grammars. *Journal of the ACM*, 16(3):423–431, 1969.

[17] P. G. Kolaitis. On the expressive power of logics on finite models. In *Finite Model Theory and Its Applications*, Texts in Theoretical Computer Science. An EATCS Series, pages 27–123. Springer Berlin Heidelberg, 2007.

[18] M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2006.

[19] L. Libkin. *Elements of Finite Model Theory*. Springer Berlin Heidelberg, 2004.

[20] P. Linz. *An Introduction to Formal Languages and Automata, Fifth Edition*. Jones & Bartlett Publishers, 2012.

[21] L. Y. Liu and P. Weiner. An infinite hierarchy of intersections of context-free languages. *Mathematical systems theory*, 7(2):185–192, 1973.

[22] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS '12, pages 101–112, 2012.

[23] M. J. Mclean and D. B. Johnston. An algorithm for finding the shortest terminal strings which can be produced from non-terminals in context-free grammars. In *Combinatorial Mathematics III*, volume 452 of *Lecture Notes in Mathematics*, pages 180–196. Springer Berlin Heidelberg, 1975.

[24] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[25] E. Moriya. A grammatical characterization of alternating pushdown automata. *Theoretical Computer Science*, 67(1):75–85, 1989.

[26] A. Okhotin. Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4):519–535, 2001.

[27] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[28] P. Sevon and L. Eronen. Subgraph queries by context-free grammars. *Journal of Integrative Bioinformatics*, 5(2), 2008.

[29] M. Tomita. An efficient context-free parsing algorithm for natural languages. In *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 2*, IJCAI'85, pages 756–764, 1985.

[30] S. H. Unger. A global parser for context-free phrase structure grammars. *Communications of the ACM*, 11(4):240–247, 1968.

[31] L. G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975.

[32] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC '82, pages 137–146, 1982.

[33] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing*, STOC '12, pages 887–898, 2012.

[34] D. Wotschke. The boolean closures of the deterministic and nondeterministic context-free languages. In *GI Gesellschaft für Informatik e. V. 3. Jahrestagung Hamburg, 8.–10. Oktober 1973*, volume 1 of *Lecture Notes in Computer Science*, pages 113–121. Springer Berlin Heidelberg, 1973.

[35] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.